Chapter 1 - Development Setup of Angular

Objectives
Key objectives of this chapter ■ Angular Files and Dependencies ■ Node.js ■ Node package manager (npm) ■ package.json ■ Semantic version numbers ■ Installing Angular ■ Application Dependencies ■ Module Loaders

1.1 Angular is Modular

- The process for downloading and adding the framework to your web application has changed completely with Angular.
- The previous version of Angular - AngularJS consisted of a single main *.js file and a few optional files:

```
angular.js
angular-route.js (optional)
```

- Angular on the other hand consists of various modules, each located in their own directory:

```
@angular\common
@angular\core
@angular\forms
@angular\http
@angular\platform-browser
@angular\router
etc.
```

- For Angular development these modules should be installed locally

1.2 Managing Angular Files and Dependencies

- Not only does Angular consist of many separate files it also relies upon various other JavaScript packages including:
  - polyfill libraries
  - module loaders
  - asynchronous programming libraries
- Downloading all of these files separately would be difficult and inefficient.
- Node Package Manager (npm) is used to simplify:
  - Downloading of Angular and related files
  - Management of local file versions
- Node Package Manager is a part of a JavaScript development platform called Node.js
- Before moving on we will review Node.js and Node Package Manager basics

1.3 What is Node.js?

- Node.js is an application development platform
- Node applications:
  - Are written in JavaScript
  - Are run from a command prompt and not in a browser
- The Node environment:
  - Is event driven
  - Is single threaded

  - Is non-blocking
  - Follows an asynchronous programming paradigm
- Many code libraries (packages) are available for Node development
- Node Package Manager (NPM) is used to install packages and manage dependencies for Node based applications
- More information is available at: https://nodejs.org

1.4 Application of Node.js

- Node.js is used to create all kinds of applications:
  - Server applications are created using the Node.js based server frameworks such as *Express*
  - Desktop Applications can be created using Node.js based desktop frameworks like *Electron* and *NW.js* (node-webkit)
  - Command line tools created with Node.js include the following:
    - Bower package manager
    - Grunt and Gulp task runners
    - Jasmine testing framework
    - Karma test runner
  - Angular web development makes use of command line tools like these as well as the npm package manager itself.

1.5 Installing Node.js and NPM

- Node and NPM are easy to install
- Windows and Mac installer packages can be downloaded from nodejs.org.

- NPM is installed along with the Node.js installation
- After installation check that node and npm are working:
  - Open a command prompt to any directory.
  - Check Node:

```
node --version
```

  - Check NPM:

```
npm  --version
```

1.6 "Hello World!" Node app

- Below is a "Hello World" application for Node.js.
- It defines a function and a variable and then calls the function.

```
// app.js file
var message = "Hello World from Node!";
function display(text){
    console.log(text);
}
display(message);
```

- The application is run from the command prompt:

```
node app.js
```

- Its output appears like this:

```
Hello World from Node!
```

- Node.js can be used like this to test select pieces of code before inserting them into web applications.

1.7 Node Libraries

- The following Node.js app uses the ***colors*** code library to output text in various colors:

```
// colorapp.js file
var color = require('colors');
var message = "Hello World from Node!";
function displayInRed(text){
    console.log(text.red);
}
displayInRed(message);
```

- Code libraries are included using the require() function

```
var color = require('colors');
```

- Many libraries are available, see:

```
https://www.npmjs.com/browse/depended
```

1.8 Node Package Manager (NPM)

- Code libraries, called packages, are installed with the npm package mgr.
- NPM uses simple commands like the following to install packages from a central repository on the web maintained by node.org:

```
npm install jquery
npm install -g gulp
```

- The ***-g parameter*** installs the specified package in a central location on the development machine. It is typically used to install large shared code libraries or node applications that include command line interfaces.
- When the -g parameter is not used packages are installed in a local sub-directory named ***node_modules***

- When npm is run without a package name it looks for a file named ***package.json*** file in the local directory that includes the required information.

```
npm install
```

- Using the package.json file multiple libraries can be installed at once

1.9 Package.json

- The package.json file includes names and versions of packages you wish to install in its ***dependencies*** section:

```
"dependencies": {
   "colors": "1.1.2",
   "lodash": "4.17.3"
},
```

- The package.json containing the above dependencies section is used to install two packages at once, the ***colors*** package and the ***lodash*** package.
- Notice how the required version number is supplied for each package.

1.10 Semantic Version Numbering

- Node Package Manager makes use of semantic version numbering.
- Semantic version numbers let you specify the exact major, minor and patch releases for a package
- Take for example the following package dependency:

```
"lodash": "4.17.3"
```

- Here the major release number is 4, the minor release is 17 and the patch release number is 3.

- Release numbers are changed for specific reasons:
  - **Major** release number are changed when a release includes "breaking" changes.
  - **Minor** release numbers are changed when new features are added while backward compatibility with earlier versions is maintained
  - **Patch** release numbers are changed when a new version includes mostly bug fixes while maintaining backward compatibility with earlier versions

1.11 Package Version Numbering Syntax

- When entering a package version number in the package.json file you can request a specific version or allow NPM to return the latest major, minor or patch release:

What you need	How to specify (example)
Exact version	2.1.5
Latest patch release	2.1 2.1.x ~2.1.0
Latest minor release	2 2.x ^2.0.0
Latest major release	*

1.12 Updating Packages

- As newer package versions are released previously downloaded versions can become obsolete.
- Use the ***outdated*** command to check if any packages have been updated since they were installed:

```
npm outdated
```

- Packages defined with an exact version number in package.json are not included in this check.
- Running the following command will bring all packages up to the latest desired version as specified in package.json:

```
npm update
```

- Updating to the latest version of a package can in some cases break your application. For this reason the update command should be used with caution.

1.13 Uninstalling Packages

- Packages no longer being used can be uninstalled using the following commands. Note thought that this does not update package.json:

```
npm uninstall package_name
```

- If you are using a package.json file and wish to uninstall a package you should:
  - Edit the package.json and remove the entry for the unused package.
  - Then running **npm prune** will remove the package from the node_modules directory

- Alternately you can uninstall a package and update the package.json at the same time using this command:

```
npm uninstall package_name --save
```

- Globally installed packages can be removed using this command:

```
npm uninstall package_name --g
```

1.14 Installing Angular Packages

- In summary, the steps to install Angular to your development machine are:
  - install node.js on your development machine using an install package from nodejs.org.
  - Create a directory for your Angular project
  - Obtain a package.json file suitable for installing Angular from an existing project or from the quickstart page on the angular.io site.
  - Copy the package.json into your project directory.
  - Open a command prompt and navigate to your project directory
  - Execute the command **npm install**
- This will create a node_modules directory and install Angular and additional dependent packages
- The Angular setup page has more details on using the Angular "Quickstart seed" project to get started

```
https://angular.io/docs/ts/latest/guide/setup.html
```

1.15 Angular CLI

- Angular also has an optional feature called Angular CLI
  - Is a command line interface for creating Angular based apps
  - Is based on Node.js and installed with NPM
  - Provides an alternative way to install Angular and develop apps.
  - Provides simple commands to create new Angular projects and add various building blocks like components and services
  - Includes a development server
  - Integrates unit tests and end-to-end testing out of the box
- For more information see:

  `https://cli.angular.io/`

Angular CLI

Although Angular CLI is certainly a useful tool, there are some things to consider when using it:

- It only uses the WebPack mnodule loader and can't be used with SystemJS (the default of the Angular Quickstart)
- It is relatively new with the final 1.0.0 release being released in March 2017 after several changes in beta and release candidate versions.
- Angular CLI greatly assists with creating new projects and defining components but becomes less useful as an application gets larger and requires manual customization anyway.
- Many tasks of Angular CLI require being online although the article below details how you might be able to work with it offline.

`http://webiks.com/working-offline-with-angular-cli/`

1.16 Angular Development Overview

- Development with Angular involves:
  - Installing Angular and dependent files
  - Creating and editing Angular code
  - Compiling typescript code files
  - Serving application files from a web server
  - Running the app in a browser
  - Debugging app code
- Moving a developed app to production typically involves:
  - Consolidating and minifying JavaScript files
  - Consolidating and minifying CSS files
  - Moving files to a production web server

1.17 Angular *Development* Dependencies

- Angular ***development*** depends on a variety of packages:
  - TypeScript compilation * (typescript)
  - TypeScript definitions (@types)
  - Testing frameworks/tools * (jasmine, karma, protractor)
  - Development server (lite-server, webpack-dev-server)
  - Module bundler (webpack)
- * Starred items are discussed in more depth later in the course

1.18 TypeScript Definitions

- TypeScript definitions for standard JS libraries are referred to as ***Typings***
- Typings provide extra information not included in standard JS libraries like:
  - interface and class definitions
  - function parameter and return types
- Typings are used to:
  - Provide code completion and documentation in programming editors
  - Verify correct usage of functions during TypeScript compilation
- Typings are typically installed by adding devDependencies in package.json like this:

```
"@types/node": "^6.0.45",
"@types/jasmine": "^2.5.35",
```

1.19 Testing Tools

- Various testing frameworks/tools designed for use with JavaScript web applications can also be used when developing Angular Applications
- The following testing tools are all Node.js based applications:
  - Jasmine: A JavaScript unit testing framework for writing tests.
  - Karma: A test runner for unit testing.
  - Protractor: An end-to-end testing framework that lets you run UI based tests in various browsers
- These tools can be installed by adding devDependencies to package.json

1.20 Development Servers

- Angular applications require files to be served from a web server and will not work properly if files are opened directly from the file system.
- Development servers:
  - Are typically written in Node.js and run directly on development machines
  - Are installed via a devDependency in package.json
  - May include auto-update features to reload pages in a browser when the underlying files change.
- Examples include:
  - lite-server,
  - webpack-dev-server
- Dedicated servers can speed up and simplify development.

1.21 Angular *Application* Dependencies

- Angular ***applications*** depend on a variety of packages
  - Runtime Module loader (systemjs)
  - Polyfills (core-js)
  - Reactive extensions (rxjs)
  - Execution contexts (zone.js)

1.22 Module Loaders

- Angular framework and application code exists in multiple files referred to as ***modules***.
- Modules are loaded as needed based on import statements like these:

```
import { NgModule } from '@angular/core';
import { AppComponent } from './app.component';
```

- Browsers don't understand modules so external packages are required to load them
- One of two methods are generally used to manage modules:
  - Load modules in the browser at run-time from separate files.

  or

  - Combine modules at compile time into a single JavaScript file that is loaded all at once by the browser.
- Packages used for module management include:
  - SystemJS - implements runtime module loading
  - WebPack - bundles modules at compile time

1.23 SystemJS Module Loader

- System JS is a run-time module loader.
- The systemjs library must be included in the application's index.html file

```
<script
src="node_modules/systemjs/dist/system.src.js">
</script>
```

- Module loading is configured via a JavaScript file:

```
<script src="systemjs.config.js"></script>
```

- Angular apps are initialized by calling the systemjs import function:

```
System.import('app')
```

- The angular.io quickstart tutorial application uses this form of module management
- More information is available at:

```
https://github.com/systemjs/systemjs
```

1.24 WebPack Module Bundler

- WebPack is a module bundler
- It is invoked during development after code files are saved and before they are loaded by the web server.
- It reads code and resolves imports by consolidating multiple module files into one or more static JavaScript assets.
- It programmatically adds script tags to the application's index.html file in order to include the consolidated JavaScript asset files.
- The modified index.html as well as the consolidated asset files can be:
  - Saved to the file system for posting to a production server
  - Saved in memory and served via the webpack-dev-server development server
- The Angular command line development tool angular-cli uses this form of module management
- More information is available at:

`http://webpack.github.io/docs/`
`https://angular.io/docs/ts/latest/guide/webpack.html`

1.25 Additional Application Dependencies

- Polyfill
  - A polyfill is code that implements required features in web browsers that don't implement the feature themselves
  - Angular makes use of the **core-js** polyfill library
  - `See: https://www.npmjs.com/package/core-js`
- Reactive Extensions
  - A library that supports asynchronous and event-based coding
  - Includes Observable objects for handling asynchronous data streams
  - Used by the Angular Http client service

    `See: https://github.com/Reactive-Extensions/RxJS`
- Execution contexts
  - Managed via the **zone.js** library
  - Used internally by Angular

    `See https://github.com/angular/zone.js/`

1.26 Summary

In this chapter we covered:

- Angular Files and Dependencies
- Node.js
- Node package manager (npm)

- package.json
- Semantic version numbers
- Installing Angular
- Application Dependencies
- Module Loaders